Local Maps: New Insights into Mobile Agent Algorithms

Bilel Derbel

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Local Maps: New Insights into Mobile Agent Algorithms

Bilel Derbel

N° 6511

April

Thème COM

ISSN 0249-6399 ISRN INRIA/RR--6511--FR+ENG

Local Maps: New Insights into Mobile Agent Algorithms

Bilel Derbel*

Thème COM — Systèmes communicants
Équipes-Projets DOLPHIN

Rapport de recherche n° 6511 — April — 23 pages

Abstract: In this paper, we study the complexity of computing with mobile agents having small local knowledge. In particular, we show that the number of mobile agents and the amount of local information given initially to agents can significantly influence the time complexity of resolving a distributed problem. Our results are based on a generic scheme allowing to transform a message passing algorithm, running on an n-node graph G, into a mobile agent one. By generic, we mean that the scheme is independent of both the message passing algorithm and the graph G. Our scheme, coupled with a well-chosen clustered representation of the graph, induces $\widetilde{O}(1)^\dagger$ ratio between the time complexity of the obtained mobile agent algorithm and the time complexity of the original message passing counterpart, while using $\widetilde{O}(n)$ mobile agents. If only k agents are allowed (k is an integer parameter), then we show that the time ratio is $O(n/\sqrt{k})$. As a consequence, we show that any global labeling function of G can be computed by exactly n mobile agents knowing their n^ϵ-neighborhood in $\widetilde{O}(D)$ time, D is the diameter of the graph and ϵ is an arbitrary small constant. We apply our generic results for the fundamental problem of computing a leader (resp. a BFS tree) under the additional restriction of $\widetilde{O}(1)$ (resp. $\widetilde{O}(n)$) bit memory per agent, and obtain $\widetilde{O}(D)$ time algorithms.

Key-words: Mobile agents, locality, local maps.

* supoprted by the DOLPHIN project team.
† $\widetilde{O}(f(n)) = \log^{O(1)} n \cdot f(n)$

Centre de recherche INRIA Lille – Nord Europe
Parc Scientifique de la Haute Borne
40, avenue Halley 59650 Villeneuve d'Ascq (France) Téléphone : +33 3 59 57 78 00 — Télécopie : +33 3 59 57 78 50

Local Maps: New Insights into Mobile Agent Algorithms

Résumé : In this paper, we study the complexity of computing with mobile agents having small local knowledge. In particular, we show that the number of mobile agents and the amount of local information given initially to agents can significantly influence the time complexity of resolving a distributed problem. Our results are based on a generic scheme allowing to transform a message passing algorithm, running on an n-node graph G, into a mobile agent one. By generic, we mean that the scheme is independent of both the message passing algorithm and the graph G. Our scheme, coupled with a well-chosen clustered representation of the graph, induces $\widetilde{O}(1)^{\ddagger}$ ratio between the time complexity of the obtained mobile agent algorithm and the time complexity of the original message passing counterpart, while using $\widetilde{O}(n)$ mobile agents. If only k agents are allowed (k is an integer parameter), then we show that the time ratio is $O(n/\sqrt{k})$. As a consequence, we show that any global labeling function of G can be computed by exactly n mobile agents knowing their n^{ϵ}-neighborhood in $\widetilde{O}(D)$ time, D is the diameter of the graph and ϵ is an arbitrary small constant. We apply our generic results for the fundamental problem of computing a leader (resp. a BFS tree) under the additional restriction of $\widetilde{O}(1)$ (resp. $\widetilde{O}(n)$) bit memory per agent, and obtain $\widetilde{O}(D)$ time algorithms.

Mots-clés : Mobile agents, locality, local maps.

‡ $\widetilde{O}(f(n)) = \log^{O(1)} n \cdot f(n)$

1 Introduction

1.1 Motivation and goals

In a mobile agent algorithm, we are given a set of mobile entities (the mobile agents) equipped with a memory and able to move from a node to another in the network in order to do some computations. The time, the number of agent moves, the number of agents, and the memory size of agents, are the complexity measures that are used most often to evaluate the efficiency of a mobile agent algorithm. The general motivation of this paper is to understand the relationship of these complexity measures, especially the *time complexity*, to the *the initial local view* complexity measure.

In order to give an intuitive definition of the *initial local view* of a mobile agent, and to show how it can interfere with the way of resolving a distributed problem, let us consider the following example: take a *non-anonymous* graph that models a set of pairwise connected locations where some robots (mobile agents) are scattered. The robots can move from one location to a neighboring one and can communicate by leaving a message in the location where they pass. Suppose that initially each robot is given a map of its surrounding locations. Suppose that the robots have to agree on some location to meet, i.e., elect a location. It is obvious that if initially the robots know the whole map of the locations (that is the entire graph), then the robots can choose the location having the smallest name to meet there. The time needed to gather the robots is then dominated by the time needed to reach that elected location. Suppose now that initially each robot is given only an incomplete or a restricted map of its environment, say a map of only the very close locations. First, it is not obvious how the information provided by the local maps can help the robots to meet. Second, assuming that the information given by the local maps does so, it is not obvious that the robots can meet as fast as if they know the whole map of their environment.

Informally speaking, the initial local view of a mobile agent measures how small is the map given to it initially. More precisely, it measures the radius of the initial map given to the agent. The purpose of this paper is to show how small local maps do help providing good solutions to classical distributed problems such as computing a leader, computing a tree, and more generally computing some labeling of a graph.

To tackle this problem independently of the distributed task, we adopt a "simulation" based approach which allows to transform any message passing algorithm into a mobile agent one. On one hand, the initial local view of a mobile agent can be interpreted from a message passing perspective as the amount of knowledge that a node may have about its neighborhood. On the other hand, many distributed problems have been shown to have efficient local solutions in the message passing setting. Thus, providing a generic and efficient simulation method will also lead to design efficient local solutions using mobile agents.

1.2 Overview of related works

To the extent of our knowledge, the relation between the initial local view as defined in this paper and the complexity of mobile agent algorithms has not been studied in previous works. However, one can find many seemingly related concepts.

The concept of *limited visibility* [FPSW05, RDW05, San01, SDY06, KA02, IKK04, FS06, AOSY99, ASM95, FNS07] is perhaps the most closely related to the issues adressed in this paper. However, the computation models used to study that concept are different from ours. This is probably due to the nature of the problems studied tehrein. In fact, existing works about the impact of limited visibility concern autonomous robots moving most often in some space, i.e., the plane. In that context, a robot is said to have a limited visibility if it can sens its surrounding up to a fixed distance, i.e., it can look and see the positions of other robots in a ball of a fixed radius at any time of the execution of the algorithm.

Another related concept, called *Oracles*, arised recently for some specific problems, e.g., broadcast / wake-up [FIP06a], tree exploration [FIP06b]. The concept of oracles is also tightly related to

the widely studied concepts of *labeling schemes* and *advice schemes* (see [FKL07, KK07, Kor]and the references there). These concepts aplly for the mobile agent model as well as for the message passing model. Informally speaking, an oracle with respect to a given problem P can be considered as an algorithm that, given a graph, outputs a labeling of the nodes in such a way solving P by a distributed algorithm becomes easier. The challenges in that context are *(i)* to design an oracle minimizing the number of bits used for the labeling of nodes, and *(ii)* to come out with a distributed algorithm that, using the information provided by the oracle (the labels), solves P efficiently. Although, studying oracles helps understanding the locality of a given problem, the issues addressed by oracles are different from those studied in this paper in many ways. In fact, paraphrasing the discussion made in [KPR02] for static labeling schemes, we can say that oracles "are centralized, in the sense that they are based on a sequential algorithm which given a description of the entire graph outputs the entire set of node labels[1]. Hence while the resulting short labels reflect local knowledge and can be used locally, their generation process is still centralized and global". Moreover, oracles are problem specific, that is the output labels are function of a fixed problem and cannot be used to resolve another one. In many cases they are also topology specific (e.g., case of trees). In this paper, we are adressing the problem of resolving any distributed task, on any network, in a fully distributed way, provided that each node knowns its surrounding up to some small distance.

From a message passing perspective, the impact of local knowledge on resolving distributed tasks has been intensively studied over the last years yielding a huge variety of results (see e.g., the seminal papers [NS95, Lin92]). The locality of distributively resolving many problems, e.g., trees, leader election, infromation dissemation, coloring, dominating sets, independent sets, partitions, tree covers, etc, can be found in the litterature. For instance, the results presented in [AGVP90] show that broadcasting in a network where each node knows its ρ-neighborhood can be done using $\Theta(\min\left\{m, n^{1+O(1)/\rho}\right\})$ messages, where m (resp. n) is the number of links (resp. nodes) in the network. The lower bound given in [KMW04, KMW06] shows, for instance, that in order to resolve the fundamental maximal independent set problem, a node must fetch some information in its $\Omega(\sqrt{\log n/\log\log n})$-neighborhood while the best deterministic upper bound solves the problem in $n^{O(1/\sqrt{\log n})}$. Reviewing all the existing results on the locality of message passing algorithms is beyond the scope of this paper. The reader is referred to [Pel00] and the pointers there for further results concerning the particulary rich state-of-the-art concerning the locality of distributed message passing algorithms. One goal of this paper is to provide a set of basic and general results that we hope will serve as a starting point to more problem specific studies of the locality level of mobile agent algorithms compared with message passing ones.

The relationship between mobile agent algorithms and message passing algorithms is studied in few works [CGMO06, BFFS03, DFSY07, FBDC99, PBD$^+$04]. In [CGMO06, BFFS03], a simulation based approach allows to provide equivalence results between tasks that can be computed with messages and those that can be computed with mobile agents. Roughly speaking, the research concern there is on determining what tasks can be computed by mobile agents and under what conditions, but not at what cost. More recently, [DFSY07] shows how to simulate message passing algorithms with mobile agents under failures with a polynomial overhead in the number of moves per agent (for simulating one message sending). That paper focuses on agent fault issues. In [FBDC99, PBD$^+$04], the authors showed how mobile agents (from a system engineering point of view) can define a *navigational* programming style for distributed parallel computing which is competitive in many points compared with classical message passing and distributed shared memory systems.

Many other investigations assuming several mobile agent models (e.g., the anonymous model) exist for specific problems such as graph exploration [DFNS05, DP04], graph learning [ABRS99, BS94], graph searching [BFNV06], decontamination [FS06], flocking [San01], gathering [KMP07, CGP06], election [BFFS07, DFNS06], etc. Few of them have considered the impact of the initial local knowledge on the complexity of mobile agent algorithms.

[1] Exceptions exist for labeling schemes but they are problem specific or topology specific, see for example [KPR02]

1.3 Main contributions

Given a (synchronous) message passing algorithm, we describe a general scheme that allows to simulate a message passing algorithm in the (asynchronous) mobile agent model (See section 2 for a detailed definition of the distributed model). Our scheme is based on partitioning the graph into a set of clusters. Roughly speaking, each cluster is controlled by some agents that simulate the message passing algorithm in that cluster while collaborating with the other agents in neighboring clusters. The computations inside each cluster is almost for free, only computing at the border of each cluster costs high.[2]

We show that the properties of the partition, namely the number of clusters, the radius of clusters, and the inter-connexion between clusters, do influence the complexity of simulating a given message passing algorithm. In particular, we establish upper bound trade-offs between the initial local view, the number of agents and the time complexity. For instance, we show that using $O(n^{1+1/\ell})$ mobile agents, the ratio between the time complexties of the original message passing algorithm and its mobile agent counterpart is $\ell^{O(1)}$ (n is the size of the graph, ℓ is an integer parameter). If the number of agents is fixed to be $k < n$, then we show that the time ratio is at most $O(n/\sqrt{k})$. Upper bounds concerning the number of agent moves are also given. The latter results hold when the initial local view of agents is $\ell^{O(1)}$ and $O(n/\sqrt{k})$ respectively.

Based on our simulation scheme, we derive a universal distributed algorithm for computing in $\widetilde{O}(D)$ deterministic time *any function* of a given graph with initially *one* mobile agent per node knowing its n^ϵ-neighborhood, $\epsilon > 0$ is any arbitrary small constant and D is the diameter of the graph. Using mobile agents with no initial information at all, the time complexity of our algorithm becomes $\widetilde{O}(\Delta + D)$ in expectation (Δ is the maximum degree of the graph). These results suggest that from a time complexity point of view, computing with mobile agents having small initial knowledge could be as powerful as message passing.

Finally, we investigate the time complexity of computing two classical and fundamental distributed problems with *limited memory* agents: the leader election and the Breadth First Spanning (BFS for short) tree. Using n mobile agents knowing their n^ϵ-neighborhood, and having respectively $\widetilde{O}(1)$ and $\widetilde{O}(n)$ bit memory, we obtain $\widetilde{O}(D)$ time algorithms. The latter two results are obtained using general techniques that apply for as well for other classical tasks, e.g., the broadcast/convergecast. Thus, one could think of desgining other sophisticated (and efficient) algorithms using only limited memory agents.

1.4 Outline

In Section 2, we define the notations and the distributed models we will consider in this paper. In Section 3, we provide two basic techniques to simulate a message passing algorithm in a mobile agent environment. These two techniques are then combined in Section 4 to provide a generic and efficient simulation scheme. In Section 5, we show how to couple our simulation scheme with some clustered representations based on sparse partitions and small dominating sets in order to obtain complexity trade-offs. In section 6, a partition algorithm (of independent interest) allowing to efficiently initialize our simulation scheme is analyzed. In section 7, we apply our results for computing any function of a given graph, then for computing a leader and a BFS tree using agents having limited memory. In section 8, we conclude the paper and give some open problems.

2 Model and Definitions

We model a network by a connected undirected graph $G = (V, E)$ where V is the set of nodes and E the set of edges. A labeled graph is a graph where nodes and edges are assigned labels.

[2]One should note that using clustered representations is not new in distributed computing. For instance, they have been used successfully in [PS96, AR93, AGLP89, KP98, Awe85, DG06, BBCD02] to find efficient distributed protocols in the message passing setting. What is new is to show how these representations can be used for studying and designing efficient mobile agent algorithms.

We denote by n, m, Δ, and D respectively the number of nodes ($n = |V|$), the number of edges ($m = |E|$), the maximum degree, and the diameter of G. We assume that:

- Each node v of G is given a unique integer identifier of at most $O(\log n)$ bits.
- The ports, that is the incident links, of a node v have distinct identifiers taken from 1 to deg_v, where deg_v is the degree of v. Notice that given an edge (u, v), the two port numbers assigned to this edge respectively for the u-leg and v-leg part are completely independent.
- The size n of the graph G is known.

We define the radius of a subgraph H of G as following: $\text{Rad}(H) = \min_{u \in H} \{\max_{v \in H} \{d_G(u, v)\}\}$, where $d_G(u, v)$ is the distance between the nodes u and v in G. Given a set of nodes C of G, we denote $G[C]$ the subgraph of G induced by the nodes of C. Given a set $\mathcal{C}$ containing some sets of nodes, we denote $\text{Rad}(\mathcal{C}) = \max_{C \in \mathcal{C}} \{\text{Rad}(C)\}$ where $\text{Rad}(C) = \text{Rad}(G[C])$. we denote $G_{\mathcal{C}} = (V_{\mathcal{C}}, E_{\mathcal{C}})$ the graph induced by $\mathcal{C}$, i.e., there is an edge between two nodes if their corresponding sets are neighbors in G. We denote by G^t the graph obtained by adding an edge between every two nodes at distance at most t in G.

In the rest of this paper, we write n^ϵ instead of $n^{O(1/\sqrt{\log n})}$. In fact, $n^{O(1/\sqrt{\log n})} = o(n^\epsilon)$ for any arbitrary small constant $\epsilon > 0$. Furthermore, since this is the first time where performance results concerning the amount of local knowledge of mobile agent algorithms are given, we omit precising the polylogarithmic overheads and use the $\widetilde{O}$ notation ($\widetilde{O}(f(n)) = \log^{O(1)} n \cdot f(n)$).

2.1 The message passing model

Each node of a graph G is an autonomous entity of computation that can communicate with its neighbors by sending and receiving messages. The output of a message passing algorithm is given by the final labeling of the graph. In the remainder, we write MsA to denote a message passing algorithm running on a graph G.

Unless stated explicitly, we concentrate on message passing algorithms that *detect the local termination*. More precisely, this assumption requires that each node *can detect that it will not make any computations no more*. For instance, we do not require a node to know that the result the algorithm is outputting was effectively computed, which is a stronger assumption. Also, we do not consider algorithms where a node knows that its label is final but it can not determine whether it will continue computing or not (for example by forwarding messages) which is a weaker assumption. Actually, our termination assumption is made only for the sake of simplicity and clarity, but fundamentally, it does not affect our results as we will point in Remarks 1, 2 and 3.

Unless stated explicitly, we consider the classical *synchronous message passing model* [Pel00]. In other words, we assume that there exists a global clock generating the same pulses for all nodes. At each pulse a node can process some messages, do some local computations and send messages to its neighbors. A message sent at a given pulse arrives before the next pulse. We denote by Time(MsA) the time complexity of algorithm MsA, that is the number of pulses from the beginning of the algorithm up to its termination. We denote by Message(MsA) the message complexity of the algorithm, that is the total number of messages exchanged by nodes.

One should remark that *synchronous* message passing algorithms are in general more efficient than their asynchronous counterparts. Thus, adapting them to the mobile agent setting is of particular interest. However, our mobile agent model is *asynchronous* as we will precise in the remainder. This is be motivated by the fact that our goal is to derive mobile agent algorithms which are both *efficient* and *independent of the synchrony* of the underlying network.

2.2 The mobile agent model

A mobile agent is a computation entity which is able to move from a node to another. We assume that:

- A mobile agent is equipped with an internal memory. Unless stated explicitly, the memory of a mobile agent is assumed to have unlimited capacity.
- Each node v has a whiteboard where agents can write and read information in a mutual exclusion manner. We assume that the whiteboards have unlimited capacity. Thus, a mobile agent can store any amount of information in a gievn whiteboard.
- When an agent arrives at node v, it is able to distinguish the edge from which it has arrived to v among all other edges of v.

One should remark that our assumptions concerning the memory capacity of the whiteboards and the agents are introduced in order to focus on the high level locality issues. However, we will take a special care to derive algorithms for many distributed tasks while using bounded memory agents.

We consider the fully *asynchronous mobile agent model*. More precisely, the mobile agent algorithms that we will describe do *not* exploit any assumptions on time such as the existence of a global clock or an upper bound on the time needed to do some actions, etc. We assume that the output result computed by a mobile agent algorithm is encoded by mean of the whiteboards of the nodes. In other words, the output is a labeling of the graph.

In the following definitions, we denote by AGA a mobile agent algorithm computing some labeling of a graph G.

Definition 1 *We say that a node is in a final state if its whiteboard will not be changed by any mobile agent. We say that a mobile agent algorithm terminates, if the algorithm ends up with all nodes in a final state.*

Definition 2 *The size of algorithm* AGA*, denoted by* Size(AGA)*, is the number of agents used by the algorithm.*

In the next defintion, we define the time complexity of a mobile agent algorithm in the *asynchronous* setting. We remark that time units in the above definition are with regard to a global, external clock, which is *not* avalaible to the agents. Hence time is observed by an "outside" observer and time units are used purely for the sake of analysis, i.e., the agent do not use any global clock to make decisions.

Definition 3 *The time complexity of algorithm* AGA*, denoted by* Time(AGA)*, is the total number of time units from the beginning of the algorithm up to its termination, assuming that an agent move induces a delay of one time unit and that the computations done by agents are time negligible.*

Definition 4 *The cost of algorithm* AGA*, denoted by* Cost(AGA)*, is the total number of agent moves from the beginning of the algorithm up to its termination.*

In the following, we will define the initial local view complexity measure. Let $\mathcal{A}$ be the set of mobile agents used by algorithm AGA. For every mobile agent $A \in \mathcal{A}$, we call h_A the homebase of agent A if at the beginning of the algorithm, agent A is at node h_A.

Definition 5 *We say that an agent $A \in \mathcal{A}$ has an* initial local view H_A*, if at the beginning of the algorithm the agent A knows a labeled connected subgraph H_A of G containing its homebase h_A.* The initial local view *of algorithm* AGA*, denoted $\mathcal{ILV}$(AGA), is then defined as the maximum radius of the initial local views of its agents. More formally,*

$$\mathcal{ILV}(\text{AGA}) \;=\; \max_{A \in \mathcal{A}} \{\text{Rad}(H_A)\}$$

We say that agent A has no initial local view if its initial local view H_A satisfies: $H_A = \varnothing$.

We remark that the latter definition formalizes the intuitive notion of local maps discussed in the introduction. However, our definition does not say whether the local maps should be stored in the memory of agents or in the memory of nodes, i.e., the whiteboards. In the rest of this paper, we will suppose that these maps are stored in the whiteboards of nodes.

In the rest of the paper, given a problem P, a message passing algorithm MsA solving P, and a mobile agent algorithm AgA sovling P, we will term *time ratio* the ratio: Time(AgA)/Time(MsA). We omit precising P, MsA and AgA when they are clear from the context.

3 Basic simulation techniques

In the remainder, we suppose that we are given a message passing algorithm MsA computing some labeling of a graph G. Our goal is to provide a mobile agent algorithm AgA that computes the same labeling.

3.1 Agents with a full initial local view

Suppose that we have only one mobile agent knowing the whole graph G, i.e., its initial local view is D. Then, algorithm MsA can be simulated in $O(n)$ time as following: First, the agent simulates algorithm MsA for each node of the graph locally at its homebase and it computes a copy of the output labeling. Second, the agent explores the graph in order to assign the final state of each node. According to our model assumptions, only the graph exploration is time consuming. Exploring a known graph can be done in $O(n)$ time, with a depth first traversal for example. Thus, any message passing algorithm can be simulated by one agent knowing the whole graph in $O(n)$ time.

Since the time complexity of the previous technique depends only on the time needed to explore the graph and mark the final states of nodes, one may ask whether we can go faster by allowing more than one agent. The answer is positive. In fact, an efficient graph structure was given in [CKR06] in order to explore a graph searching for black holes. That data structure represents the graph using a forest of k trees (spanning the whole graph), each tree has at most $O(n/k)$ nodes. Thus, using k mobile agents, the agents can mark the nodes of the graph in parallel, i.e., each agent is assigned one tree, each agent traverses the graph using a shortest path leading to that tree, and each agent traverses the assigned tree to mark its nodes with their final states.

Using the *"balanced tree"* structure of [CKR06], the following holds:

Theorem 1 *Given an integer parameter $k \geqslant 1$, any labeling function of G can be computed by an asynchronous mobile agent algorithm* AgA *such that:*

$\mathcal{ILV}$(AgA)	D
Size(AgA)	k
Time(AgA)	$O(n/k + D)$
Cost(AgA)	$O(n + k \cdot D)$

3.2 Agents with no initial local view

The one-node-one-agent technique: Suppose that each node is assigned a mobile agent having no initial local view. Then, a simple idea to simulate algorithm MsA is to make each agent simulate the instructions that the algorithm would have done at his homebase. For that purpose, we classify the instructions of algorithm MsA into two types:

(i) *A send instruction*: a node u sends a message to a neighbor v.

(ii) *A compute instruction*: a node u makes some computation, e.g., it checks whether it has received a message, it updates the value of its variables, it waits until the reception of a message from a given neighbor.

In order to simulate the instructions of algorithm MsA in node u by the mobile agent A_u assigned to u, let us first consider the following technique:

- Using the whiteboard of u, agent A_u creates deg_u *receive-buffers* corresponding to the ports of u. Each receive-buffer is identified by its corresponding port.
- To simulate a send instruction of a message from node u to a node v, agent A_u stores the message in its internal memory. Then, it crosses the edge connecting u to node v. Once at node v, it writes the message in the corresponding receive-buffer of the whiteboard of v. After that, the agent goes back to u.
- To simulate a compute instruction, agent A_u makes the same computation using the whiteboard of u.

Clearly, the previous technique is correct when algorithm MsA is asynchronous. However, it may fail when the algorithm is synchronous. Typically, that may happen if the decisions made by the algorithm are based on the pulse numbers of the global clock. Furthermore, the previous technique may fail even if we assume that the mobile agent model is synchronous.

A solution that solves the problems due to synchrony is to use a synchronizer-α like algorithm [MS00, Awe85] in order to "synchronize" an agent A_u with the other agents in the neighborhood of u. More precisely, in order to simulate the send instructions of a given pulse p, each agent A_u proceeds in two stages. At the first stage, agent A_u simulates the send instructions as explained before. At the second stage, the agent moves sequentially to each node v in its neighborhood and it writes a special *<IamSafe in p>* message in the corresponding receive buffer of v saying that it has finished simulating the instructions of pulse p. When the agent A_u learns that all the agents in its neighborhood are also safe, then it can proceed with pulse $p+1$ and so on. Clearly, simulating one pulse is at most $O(\Delta)$ time consuming, and requires the agents to move $O(m)$ times.

The master/worker technique: Now, suppose that at each node u there is initially $deg_u + 1$ agents. Among these agents, suppose that there is a distinguished agent called *master* and deg_u agents called *workers*. Thus, to simulate a send instruction, the master can delegate this work to one worker. The communication between the masters and their workers can be easily implemented using the whiteboards of nodes. One can see that the time ratio is then reduced to $O(1)$. This simple technique is efficient from a time complexity point of view but it is agent consuming, i.e., we need $O(m)$ agents. In the next section we will show how to exploit the master/worker technique to get better performances.

The performances of the previous techniques are resumed by the following theorem:

Theorem 2 *Any synchronous message passing algorithm* MsA *can be simulated by an asynchronous mobile agent algorithm* AgA *such that:*

$\mathcal{ILCV}$(AgA)	0	
Size(AgA)	n	$O(m)$
Time(AgA)	$O(\Delta \cdot \text{Time(MsA)})$	$O(\text{Time(MsA)})$
Cost(AgA)	$O(m \cdot \text{Time(MsA)})$	

4 A generic simulation scheme

In this section, we give a generalization of the basic simulation techniques described in the previous section. The idea of the generalization is to combine those techniques with a clustered representation of the graph. We will see that our generalization leads to new trade-offs between the complexity measures.

In the reaminder of this section, we will consider the following hypothesis:

> **H**($\mathcal{C}$): *We are given a precomputed partition $\mathcal{C}$ of the graph G into disjoint regions. Each region C has a distinguished node r_C called the center and a precomputed spanning tree T_C rooted at the center. At node r_C, there is a mobile agent called the master, together with some other workers (eventually none). Each master knows a labeled copy of the neighborhood of its region. In particular, he knows the edges leading to different regions.*

For the clarity of the presentation, distributed methods to cope with hypothesis **H**($\mathcal{C}$) will be presented later in sections 5 and 6. In particular, we will show how to construct a partition $\mathcal{C}$ and how to initialize the masters and the workers by efficient distributed algorithms. For now, we concentrate on describing and analysing our generic simulation scheme under the hypothesis **H**($\mathcal{C}$).

4.1 Description of the scheme

Provided that **H**($\mathcal{C}$) is true, any synchronous message passing algorithm MsA can be simulated by the generic scheme described in Fig. 1 below.

There are two key points in our scheme:

- The messages sent by nodes in a region C to nodes in a region C' ($C \neq C'$) are concatenated into only one message $M_{C \to C'}$ (Step 1.*c*). The message $M_{C \to C'}$ is delivered to the master in center node $r_{C'}$ by one worker at once, thus avoiding to deliver the messages one by one.
- The pulses of algorithm MsA are simulated using a combination of a synchronizer-γ like [MS00, Awe85] algorithm and a synchronizer-α like algorithm. Roughly speaking, the master synchronizes the nodes in its region by itself (This is made possible since the master knows the entire topology of its region). Then the master (using its workers) synchronizes its region with neighboring ones using the <*IamSafe*> messages.

Hereafter, we give more details about the low level (yet important) distributed problems raised by our high level description and the way to resolve them:

- How a worker agent at node r_C knows the route leading to a center node $r_{C'}$ (Steps 1.*b* and 1.*c*)? This can be solved using the spanning trees T_C and $T_{C'}$ (remember that our scheme assumes that **H**($\mathcal{C}$) is satisfied). In fact, the master knows an inter-region edge leading to C'. Hence, the master can tell the worker which path in T_C leads to that edge. Once arrived in C' the worker uses the rooted tree $T_{C'}$ to reach the center $r_{C'}$. It also uses its internal memory to remember the route it took to reach $r_{C'}$ and thus to go back home to r_C.
- How the concatenated messages are delivered (Step 1.*c*)? This is done by adding the identifiers of the sender and the receiver to each message.
- How a master simulates a compute instruction in the form: "wait upon the reception of a message from ..."? This type of instruction is transformed in the form: "while no message has been received from ... then check for the message at next pulse". Thus, a master agent is never blocked.

Input: a graph G with $\mathbf{H}(\mathcal{C})$ satisfied; a synchronous message passing algorithm MsA.

Output: algorithm AgA in a region C.

1. For each pulse p in algorithm MsA, the master agent of a region C simulates the instructions of pulse p for all nodes in C (locally in r_C):
   (a) For every node in C, the master creates a collection of receive-buffers in the whiteboard of r_C.
   (b) To simulate a send instruction from node $u \in C$ to node $v \in C$, the master writes the message in the corresponding receive-buffer created in r_C.
   (c) *If some nodes in C send some messages in algorithm* MsA *to some other nodes belonging to a region $C' \neq C$, then the master concatenates those messages into only one message. Then, one worker is chosen to deliver that message to $r_{C'}$. The worker returns to r_C once its job is finished.*
   (d) If a node $u \in C$ makes some computations in algorithm MsA, then the master of C makes the same computations using the withebord of r_C.
   (e) Once the master finish simulating the sending of messages of pulse p, the master synchronizes with other masters in neighboring region: the workers move to neighboring regions and deliver a special $<$*IamSafe* $(C, p)>$ message.
   (f) Once the master is safe and learns that all the neighboring masters are safe, then it proceeds with the next pulse $p + 1$.
2. Once a master and its workers have finished simulating the instructions of algorithm MsA in C:
   (a) They inform other masters in neighboring regions that the simulation is terminated for region C by delivering an $<$*IamDone* $C>$ message.
   (b) They continue synchronizing with a neighboring master in region C' until an $<$*IamDone* $C'>$ message is delivered by that master.
   (c) They mark every node in C with its final state computed in Step 1. (Once an $<$*IamDone*$>$ message is delivered by each neighboring master).

Figure 1: The simulation scheme.

- What is the $<$*IamDone*$>$ messages in Step 2? These messages allow us to terminate the mobile agent algorithm correctly. In fact, if we allow a master agent to stop computing just after the simulation of algorithm MsA is finished in its regions, then it may happen that a neighboring master waiting for a synchronization message get blocked. Remark that in our scheme a master agent continues synchronizing until all the neighboring masters terminate the simulation in Step 1.
- When the simulation of algorithm MsA does terminate (Step 2)? Because we only consider message passing algorithm that detect the local termination, the master can detect the local termination of each node in its region and thus it knows when to execute the last step of Fig 1. The case of weaker message passing termination assumptions is treated in Remarks 1, 2 and 3.

4.2 Complexity analysis

Clearly, the simulation of a send instruction between two nodes *not* in the same region dominates the overall time complexity per pulse. The concatenation mechanism of our scheme allows to deliver the messages exchanged by nodes in a region C and nodes in a different region C' at once.

Therefore, assuming that each master agent has as many workers as neighboring regions, it takes at most $2\text{Rad}(C) + 2\text{Rad}(C') + 1$ time to simulate all the send instructions of a given pulse of algorithm MsA. Thus, by using at most $|\mathcal{C}|$ master agents plus $2\,|E_{\mathcal{C}}|$ workers, each pulse can be simulated in $O(\text{Rad}(\mathcal{C}))$ time. As for the cost complexity, one can see that simulating each pulse requires the workers to deliver the messages of the original algorithm and to synchronize with neighboring masters. Thus, the cost complexity is $O(\text{Rad}(\mathcal{C}) \cdot |E_{\mathcal{C}}|)$ per pulse.

After finishing the simulation of the pulses of algorithm MsA, the nodes must be marked with their final states (Step 2.*c*). This step has also a time and a cost complexity. In the following, we denote by $\tau(\mathcal{C})$ the maximum time needed for each master agent (and its workers) to mark the whiteboards of nodes in its region with their final states. We also denote by $\eta(\mathcal{C})$ the cost complexity of marking the nodes with their final states, that is the total number of moves that all agents make in order to mark the whiteboards of all nodes in the graph. The analysis of $\tau(\mathcal{C})$ and $\eta(\mathcal{C})$ is delayed to section 5.

Lemma 1 *Under the hypothesis* $\mathbf{H}(\mathcal{C})$*, any synchronous message passing algorithm* MsA *can be simulated by an asynchronous mobile agent algorithm* AgA *such that:*

$\mathcal{ILV}$(AgA)	Rad($\mathcal{C}$)
Size(AgA)	$O(\|\mathcal{C}\| + \|E_{\mathcal{C}}\|)$
Time(AgA)	$O(\text{Rad}(\mathcal{C}) \cdot \text{Time}(\text{MsA})) + \tau(\mathcal{C})$
Cost(AgA)	$O\left(\text{Rad}(\mathcal{C}) \cdot \|E_{\mathcal{C}}\| \cdot \text{Time}(\text{MsA})\right) + \eta(\mathcal{C})$

Remark 1 *Suppose that we have a message passing algorithm that does not detect the local termination, e.g., at least one node can not say whether it has finished the computations. Then, we can modify our simulation scheme so that,* at each round *a master and its workers mark the nodes in their region with the labels computed by the message passing algorithm, instead of waiting the end of the simulation. Thus, at each round, the labeling of nodes will be the same than in the message passing algorithm. This simple modification allows to obtain a termination property for our mobile agent scheme which is 'similar' to that of the original message passing algorithm. Of course, this is achieved at the price of increasing the time and the cost complexities, i.e., the overhead will be order of* $\tau(\mathcal{C})$ *and* $\eta(\mathcal{C})$ *(respectively) at each round. As we will discuss in Remarks 2 and 3, the overhead can be made negligible compared to the complexity of the simulation itself.*

4.3 Synchrony issues

First, remark that if we consider the *synchronous* mobile agent model, then we can prove that the cost complexity is at most $O(\text{Rad}(\mathcal{C}) \cdot \text{Message}(\text{MsA}))$ or $O(\text{Rad}(\mathcal{C}) \cdot |E_{\mathcal{C}}| \cdot \text{Time}(\text{MsA}))$, plus $\eta(\mathcal{C})$, without time penality.

Second, we remark that a correct asynchronous message passing algorithm should work when the delay induced by a message is arbitrary. In other words, the only assumption we made is that the messages sent by a node arrive to destination. Therefore, by simply removing the synchronization mechanism from the previous synchronous simulation scheme, we can obtain a new simulation scheme for asynchronous message passing algorithms. Of course, such an asynchronous scheme will have different performance measures. One can prove that the cost complexity is at most $O(\text{Rad}(\mathcal{C}) \cdot \text{Message}(\text{MsA}))$. The best upper bound on the time complexity we can obtain is also $O(\text{Rad}(\mathcal{C}) \cdot \text{Message}(\text{MsA}))$. In fact, in the general case of *any* asynchronous message passing algorithm, it is no longer possible to avoid delivering the messages of the algorithm one by one using the concatenation mechanism. Nevertheless, we can prove that for many classes of asynchronous message passing algorithms, the concatenation mechanism will work efficiently and leads to the same time ratio as in the synchronous case. Since we are essentially interested in the time complexity, we omit the detailed analysis.

5 Coupling our scheme with efficient clustered representations

The efficiency of our simulation scheme depends on the properties of the partition $\mathcal{C}$ we are given. For instance, in order to reduce the number of agents, we should use a partition of the network into few regions connected by few edges. If we want to reduce the time ratio, then we should use a partition having a small radius. If we want to reduce the cost complexity, then we should use a partition minimizing the radius and the inter-region edges. Finding a partition having a good compromise between the three criteria, radius, inter-region edges, and number of regions is difficult. However, there exist partitions with a good compromise between only two criteria. Furthermore, we can construct these partitions distributively by efficient algorithms.

5.1 Sparse partition based scheme

Sparse decompositions [AP90] allow to represent a graph by a set of clusters with a good compromise between the radius of the clusters and the number of inter-cluster edges. In Section 6, we will prove the following lemma:

Lemma 2 *In the message passing model, there exists a deterministic (resp. randomized) distributed algorithm that given an n-node graph G and an integer parameter k, constructs a partition structure $\mathcal{C}$ such that* $\mathrm{Rad}(\mathcal{C}) = O(k^c)$ *and* $|E_{\mathcal{C}}| = O(n^{1+1/k})$ *in* $k^c \cdot n^\epsilon$ *time (resp.* $O(k^c \cdot \log n)$ *expected time) where c is a constant ($c = \log_2 5$).*

Suppose we use the partition of the previous lemma at the bottleneck of our generic scheme. What upper bounds can we give for $\tau(\mathcal{C})$ and $\eta(\mathcal{C})$? Using at most $O(n)$ agents, it is easy to see that $\tau(\mathcal{C})$ and $\eta(\mathcal{C})$ can be bounded by $O(\mathrm{Rad}(\mathcal{C}))$ and $O(\mathrm{Rad}(\mathcal{C}) \cdot n)$ respectively (Remember that the masters know the topology of their regions).

Now, applying Theorem 2 in the context of the sparse partition algorithm given by Lemma 2, one can derive a preprocessing mobile agent algorithm constructing the required partition. In the following, we will denote by PREPROCESS_PART such an algorithm.

From the discussion above, and combining Lemma 2 and Lemma 1, the following holds:

Theorem 3 *Given an integer parameter k, any graph G can be preprocessed by an asynchronous mobile agent algorithm* PREPROCESS_PART *verifying*[3]*:*

$\mathcal{ILV}$(PREPROCESS_PART)	0
Size(PREPROCESS_PART)	n
Time(PREPROCESS_PART)	$k^c \cdot \Delta \cdot n^\epsilon$
$\mathbb{E}$ (Time(PREPROCESS_PART))	$k^c \cdot \Delta \cdot \log n$

such that after the preprocessing phase, any synchronous message passing algorithm MSA *can be simulated by an asynchronous mobile agent algorithm* AGA *verifying:*

$\mathcal{ILV}$(AGA)	$O(k^c)$
Size(AGA)	$O(n^{1+1/k})$
Time(AGA)	$O(k^c \cdot \mathrm{Time}(\mathrm{MSA}))$
Cost(AGA)	$O\left(k^c \cdot n^{1+1/k} \cdot \mathrm{Time}(\mathrm{MSA})\right)$

Remark 2 *Using Remark 1, the time (resp. cost) overhead needed to handle other termination assumptions on algorithm* MSA *can be proved to be $O(k^c)$ (resp. $O(k^c \cdot n)$) per round which is negligible.*

[3]The constant c in the following tables is the same than in Lemma 2.

5.2 Dominating set based scheme

In this section, we want to fix the number of agents used by our simulation scheme to be a parameter $k < n$. For that purpose, we use a graph structure based on small dominating sets. A ρ-dominating set S of G is a set of nodes verifying: $\forall v \in V$, $\exists s \in S$ such that $d_G(v, s) \leqslant \rho$. Given a ρ-dominating set of G, a partition of G can be obtained by clustering the nodes of the graph around the nodes of the dominating set. Based on the results presented in [KP98], we have:

Lemma 3 ([KP98]) *In the message passing model, there exists a deterministic distributed algorithm that given an n-node graph G and a parameter ρ ($< n$) constructs a partition structure $\mathcal{C}$ such that $\mathrm{Rad}(\mathcal{C}) = O(\rho)$ and $|\mathcal{C}| = O(n/\rho)$ in $O(\rho \cdot \log^* n)$ time.*

Applying Lemma 1 in the case of a dominating set based partition is less straightforward. First, the best upper bound we can give to $|E_{\mathcal{C}}|$ is $O(|C|^2)$. Hence, to obtain a total of k agents, we use at most $\sqrt{k}$ master agents, each one has at most $\sqrt{k}$ workers, i.e., we have to chose parameter $\rho = n/\sqrt{k}$. Second, when the number of agents is fixed, we have to bound $\tau(\mathcal{C})$ and $\eta(\mathcal{C})$. Using a trivial traversal to mark the final states of nodes will lead to upper bounds that dominate the complexity of the simulation itself. An efficient solution is to use the algorithms of Theorem 1. In other words, we use the balanced tree structure of [CKR06] inside each region of the partition. Thus, using only $\sqrt{k}$ agents per region of radius $O(n/\sqrt{k})$ each, we can bound $\tau(\mathcal{C})$ and $\eta(\mathcal{C})$ by $O(n/\sqrt{k})$ and $O(n)$ respectively.

By applying Theorem 2 to the partition given by Lemma 3 we can to derive a preprocessing algorithm called PREPROCESS_DOM, such that the following holds:

Theorem 4 *Given an integer parameter $k < n$, any graph G can be preprocessed by an asynchronous mobile agent algorithm* PREPROCESS_D *verifying:*

$\mathcal{ILV}$(PREPROCESS_DOM)	0
Size(PREPROCESS_DOM)	n
Time(PREPROCESS_DOM)	$O\left(\Delta \cdot n/\sqrt{k} \cdot \log^* n\right)$

such that after the preprocessing phase, any synchronous message passing algorithm MSA *can be simulated by an asynchronous mobile agent algorithm* AGA *verifying:*

$\mathcal{ILV}$(AGA)	$O\left(n/\sqrt{k}\right)$
Size(AGA)	k
Time(AGA)	$O\left(n/\sqrt{k} \cdot \mathrm{Time(MSA)}\right)$
Cost(AGA)	$O\left(\sqrt{k} \cdot n \cdot \mathrm{Time(MSA)}\right)$

Remark 3 *Based on Remark 1, one can prove that the time/cost overhead needed to handle other termination assumptions is negligible when using only k mobile agents.*

6 Efficient distributed initialization of the simulation scheme

In the previous section, we did not care about how the agents are initialized distributively after the preprocessing phase, i.e., how the masters and the workers are initialized distributively. In the following paragraphs, we will give efficient distributed algorithms proving that the complexity of initializing the agents is negligible.

6.1 Basic observations

Suppose that initially we have one agent per node. After the preprocessing of Theorem 3 or Theorem 4, each mobile agent can say whether it is a center of a region or not. Thus, it is easy to initialize the master agents: Each mobile agent who identifies its homebase as the center of a region becomes the master of its region. A trivial solution to initialize the workers is to allow the master to create as many workers as needed (such a clone like [AMZ06, FS06] solution may be acceptable in practice). In the following, we will *not* make such an assumption.

A first idea to initialize the workers is to let an agent whose homebase is not a center of a region be a worker in its region. This idea will work in the case of Theorem 4. In fact, the ρ-dominating set algorithm of [KP98] also allows to partition the graph into regions having at least n/ρ nodes (This non trivial property is proved in [KP98]). Hence, if each agent in a non-center node becomes worker and joins his master (at distance $\rho = n/\sqrt{k}$), then each master agent will have the required number of workers assumed implicitly in Theorem 4 (The hypothesis $\mathbf{H}(\mathcal{C})$ is handled using the labeling computed by workers). Thus, the complexity of initializing the agents is negligible compared to the preprocessing.

The same idea will not work in the case of Theorem 3 since the number of required workers could be $\Theta(n^{1+1/k})$. One could think that if initially there are $\Theta(n^{1/k})$ mobile agents per node, then the previous initialization technique would hold. This is not true since the maximum degree of the graph $G_{\mathcal{C}}$, where $\mathcal{C}$ is the partition constructed in Lemma 2, is not bounded by $O(n^{1/k})$. Partitions having small maximum degree exist. But, to our knowledge, there are no efficient distributed algorithms to construct them (By efficient here we mean having running time n^{ϵ}).

Hence, the question is : assume that we have $\Theta(n^{1/k})$ agent per node, how can we initialize the workers needed for Theorem 3 efficiently ? In the following, we give an answer to this question using algorithm PART described and analyzed in the next paragraphs.

6.2 Algorithm Part

Algorithm PART is given in Fig. 2 (The MIS procedure of Step 2.*c* denotes a Maximal Independent Set procedure [Pel00]).

The idea of algorithm PART is to construct the partition of Lemma 2 and at the same time to chose some particular edges, called *preferred* edges, that will help us initializing the workers. We remark that algorithm PART is an adaptation of algorithm STRATEGY$_2$ of [DG06] (algorithm PART allows to improve the size of the spanners constructed there by a $\log k$ factor which could be of independent interest).

The proofs of the two following lemmas are essentially the same than the correctness proof of algorithm STRATEGY$_2$ of [DG06]. Hence, we will only give a sketch.

Lemma 4 *The radius of the partition $\mathcal{C}$ constructed by algorithm* PART *is at most $k^{O(1)}$.*

Proof.(sketch) Remark that the size of a merged region increases exponentially each iteration. Thus, after at most $\log k$ iterations all the regions will satisfy the sparsity condition of Step 2.*a*. The lemma is now proved by showing that at each iteration the radius of each new merged region increases by at most a multiplicative factor $O(1)$. □

Lemma 5 *In the message passing model, algorithm* PART *can be implemented by a deterministic (resp. randomized) distributed algorithm in $k^{O(1)}\ n^{O(1/\sqrt{\log n})}$ (resp. $k^{O(1)} \cdot \log n$) time.*

Proof.(sketch) Use the best known MIS algorithms [Pel00] □

In the next lemma, we will state some properties concerning the set of preferred edges contructed by algorithm PART.

First, let us remark that the definition of "an outgoing edge with respect to" a region R is given in Step 2.*e* of algorithm PART. Roughly speaking, it means that the edge is oriented from

Input: An n-node graph G, and an integer $k \geqslant 1$.

Output: a partition $\mathcal{C}$ and a set of preferred edges $\overrightarrow{\mathcal{P}}$.

1. $\mathcal{C} := \varnothing$; $\mathcal{R} := V$; $\forall v \in V$, $R(v) := \{v\}$, $\overrightarrow{\mathcal{P}} = \varnothing$.
2. **For** $i = 1$ **to** $\log k + 1$ **do:**
   (a) Let $\mathcal{L} := \left\{R \in \mathcal{R}, \quad |\mathcal{N}_{G_\mathcal{R}}(R)| \leqslant n^{1/k}\,|R|\right\}$, where $|\mathcal{N}_{G_\mathcal{R}}(R)|$ is the number of the neighboring regions of R in the graph $G_\mathcal{R}$ induced by the regions in $\mathcal{R}$..
   (b) Let $\mathcal{X} := \mathrm{MIS}((G_\mathcal{R})^2[\mathcal{R} \setminus \mathcal{L}])$.
   (c) If a region $R \in \mathcal{R} \setminus \mathcal{X}$ is at distance at most 2 from a region of set $\mathcal{X}$ in the graph $G_\mathcal{R}$, then merge R with its closest region $X \in \mathcal{X}$ breaking ties using region identifiers (the identifier of a region is the identifier of its center. The center of the new merged region is the center of X.)
   (d) Let $\mathcal{M}$ be the set of new merged regions, and $\mathcal{U}$ the set of unmerged regions.
   (e) $\forall R \in \mathcal{U}$, if R has a neighboring region R' in $\mathcal{M}$ or in $\mathcal{U}$ then R selects one preferred edge $\overrightarrow{e}$ leading to R'. The preferred edge $\overrightarrow{e}$ is called an *outgoing edge with respect to* R. Update $\overrightarrow{\mathcal{P}} := \overrightarrow{\mathcal{P}} \cup \{\overrightarrow{e}\}$.
   (f) $\forall R \in \mathcal{C}$, if R has a neighboring region $R' \in \mathcal{M}$ then let $\mathcal{E}(R \to R') = \{\overrightarrow{e_1}, ..., \overrightarrow{e_\ell}\}$ be the subset of outgoing edges with respect to R that lead to R'. If $\ell \geqslant 2$, then update the set $\overrightarrow{\mathcal{P}}$ by removing all but one edge from $\mathcal{E}(R \to R')$.
   (g) Update $\mathcal{R} := \mathcal{M}$ and $\mathcal{C} := \mathcal{C} \cup \mathcal{U}$.
3. $\forall R \in \mathcal{C}$, construct a BFS tree rooted at its center.

Figure 2: Algorithm Part

the region R to outside R. For instance, take an outgoing edge $\overrightarrow{e} \in \overrightarrow{\mathcal{P}}$ with respect to R that leads to R'. Consider the endpoints $u \in R$ and $v \in R'$ of that edge. Then, the edge (u, v) may also correspond to an outgoing edge $\overrightarrow{e'}$ with respect to R'. But in this case, $\overrightarrow{e}$ and $\overrightarrow{e'}$ define two different edges in set $\overrightarrow{\mathcal{P}}$. Also, we remark that if $\overrightarrow{e}$ is an outgoing edge with respect to R that leads to R' then, it does not necessarily mean that there exists an outgoing edge with respect to R' that leads to R.

Lemma 6 *The set of preferred outgoing edges $\overrightarrow{\mathcal{P}}$ computed by algorithm* Part *satisfies:*

- *For every region $R \in \mathcal{C}$, there are at most $n^{1/k}\,|R|$ outgoing edges with respect to R.*
- *For every region $R \in \mathcal{C}$, and for every two outgoing edges with respect to R and leading to regions R' and R'', we have $R' \neq R''$.*
- *For every $R, R' \in \mathcal{C}$, there exists at least one edge $\overrightarrow{e} \in \overrightarrow{\mathcal{P}}$ connecting R and R'.*

Proof.(sketch) The outgoing edges with respect to a region R are added only at one iteration where R falls into set $\mathcal{U}$ (Step 2.*e*). The first claim of the lemma follows by remarking that $\mathcal{U} \subset \mathcal{L}$, and the number of neighbors of R in set $\mathcal{M}$ or set $\mathcal{U}$ is at most the number of neighbors of R before the merge process of Step 2.*c*.

The two other claims follow by construction from Step 2.*c*. and Step 2.*f*. □

Remark 4 *We remark that the correctness of Lemma 2 is a straightforward consequence of the two previous lemmas.*

Let $\mathcal{A}_u$ be the set of mobile agents that are initially in a node u. Then, using algorithm PART, we can initialize the masters and the workers as following:

- If node u is a center of a region R, then one agent from the set $\mathcal{A}_u$ becomes the master of R. The others become workers in R.
- If a node u is not a center, then u must belong to a region R with center v. Hence, all the agents of the set $\mathcal{A}_u$ join the master at node v using the BFS tree of R and become workers in R. Once arrived at v, they also inform the master about the outgoing edges with respect to R that are incident to their homebase u.
- For each outgoing edge $\overrightarrow{e}$ with respect to a region R and leading to a region R', the master of R asks one worker to join the master of R' and to be worker there.

By Lemmas 4 and 6, and using the previous initialilzation technique, $2n^{1/k}+1$ agents per node are sufficient to initialize the masters and the workers in $k^{O(1)}$ time (and cost) after the preprocessing of Theorem 3, which is negligible.

7 Applications

In this section we will concentrate on the time complexity and omit discussing the cost complexity.

7.1 On computing any labeling of a graph

First, any labeling function of G can be computed by a message passing algorithm in $O(D)$ time in the message passing model (by collecting the topology of G at one node, computing the labeling locally and broadcasting the result).

Second, if we have a task which can be computed by a deterministic message passing model in t time, then all the information used by a node is in its t-neighborhood. Thus, if we have a mobile agent per node, and if the mobile agents know the $O(t)$-neighborhood of their homebases, then each mobile agent can construct in $O(1)$ time a labeled copy of its $O(t)$-neighborhood where the labels assigned to nodes and edges of that neighborhood copy correspond to the final labeling computed by the message passing algorithm.

For $k = \log n$, we have $n^{1/k} = O(1)$. Hence, using $O(1)$ agents per node allows us to run algorithm PART and also to initialize the masters and the workers efficiently. Thus, from the previous discussion we can derive an efficient algorithm for computing any labeling of a graph using $O(1)$ agents per node. In the following, we will argue that *assuming that initially there is exactly one agent per node* leads to essentially the same complexity trade-offs.

Theorem 5 *Given any graph G with one mobile agent assigned to each node, any labeling function of G can be distributively computed by a deterministic (resp. randomized) asynchronous mobile agent algorithm* DET_AGA *(resp.* RAND_AGA*)*[4] *verifying:*

$\mathcal{ILV}$(DET_AGA)	0	n^ϵ
Time(DET_AGA)	$n^\epsilon \cdot \Delta + \widetilde{O}(D)$	$\widetilde{O}(D)$

$\mathcal{ILV}$(RAND_AGA)	0
$\mathbb{E}$(Time(RAND_AGA))	$\widetilde{O}(\Delta + D)$

[4]We have given the names DET_AGA and RAND_AGA to detyerministic and randomized algorithms in order to clarify the presentation and to ease the comparaison with previous results. However, these two algorithms are not explicitly described in the paper as they can be implicitly derived from the generic simulation technique and the proof of the theorem.

Proof. Assume that there is one agent per node and take $k = \log n$ in algorithm PART. Every agent simulates algorithm PART locally. If the homebase is not a center, then the agent becomes a worker in its region, otherwise it is a master. Thus the master of a region R owns $|R| - 1$ workers.

Consider two neighboring regions R and R' computed by algorithm PART.

Let $\overrightarrow{\mathcal{R}}$ be the set of regions that are incident to the outgoing edges with respect to R. By Lemma 6, we have $|\overrightarrow{\mathcal{R}}| = O(1) \cdot |R|$. Notice that the set $\overrightarrow{\mathcal{R}}$ does *not* define *all* the neighboring region of R. By Lemma 6, one can see that we have three cases: *(i)* $R' \in \overrightarrow{\mathcal{R}}$ and $R \notin \overrightarrow{\mathcal{R}'}$, or *(ii)* $R' \notin \overrightarrow{\mathcal{R}}$ and $R \in \overrightarrow{\mathcal{R}'}$, or *(iii)* $R' \in \overrightarrow{\mathcal{R}}$ and $R \in \overrightarrow{\mathcal{R}'}$.

Remark that at each pulse (of the original message passing algorithm), at most one concatenated message must be delivered from R to R', and symmetrically at most one concatenated message must be received by R from R'.

Thus, to simulate the instructions of a message passing algorithm at a given pulse, the idea is to make the workers of each region R deliver the concatenated messages to the regions in $\overrightarrow{\mathcal{R}}$ in at most $O(1)$ stages. Each stage requires that the $|R| - 1$ workers traverse their region R, plus an additional neighboring region. Once arrived at the center of a neighboring region, a worker delivers a concatenated message, and it also asks the master there whether any concatenated message must be delivered to R. If any, then the worker goes back home with that concatenated message. If none, then the worker also goes back home. However, by using only this technique we may miss delivering some concatenated messages. In fact, take two regions R and R' satisfying: $R' \in \overrightarrow{\mathcal{R}}$ and $R \notin \overrightarrow{\mathcal{R}'}$, and suppose that at a given pulse, there exists *no* concatenated message to deliver from R to R' but there do exist a concatenated message $M_{R' \to R}$ to deliver from R' to R. Since, no worker will move to the center of R', the concatenated message $M_{R' \to R}$ is missed. Note also that this is the only case where a concatenated message may be missed. To handle this case, we make the workers systematically move to all the neighboring regions defined by $\overrightarrow{\mathcal{R}}$ and ask whether a concatenated message has to be delivered back to R. Thus, each pulse of the original algorithm is simulated in at most $O(1)$ stages, each stage lasts at most $\widetilde{O}(1)$ time.

Synchronizing the masters in neighboring regions can also be handled using the same technique. □

Remark 5 *The previous results should be compared with the fact that for any graph G, there exists a labeling function of G that can not be distributively computed by an algorithm using $O(n)$ agents having no initial local views in time better than $\Omega(D)$ (e.g., compute a tree, a leader) and $\Omega(m/n)$ (e.g., compute a copy of the graph).*

7.2 On computing with small memory agents

In previous sections, we have always considered that agents are equipped with enough memory when simulating the instructions of a given message passing algorithm. Actually, only the workers need to have enough internal memory to simulate message sending from a region to another. In fact, since a master never moves from its center node, then it can use the memory provided by the whiteboard to store the information needed for the simulation. However, a worker has to *(i)* store the concatenated messages (and the synchronization messages), *(ii)* store the route used to go from a center node to another one, and *(iii)* store the information needed to mark nodes with their final labeling at the end of the simulation:

- The size of the concatenated messages depends both on the size of the messages in the original algorithm to simulate and on the number of edges connecting the nodes of two neighboring regions. In the worst case, $\widetilde{O}(m \cdot b)$ bits of memory per agent are needed, where b is the maximum size (in bits) of a message sent by the original algorithm. The synchronization process requires only $\widetilde{O}(1)$ bit memory.

- Using Theorem 3 for $k = \log n$, two center nodes are at distance at most $\widetilde{O}(1)$. Thus, the memory needed by a worker to store a shortest path between two center nodes is at most

$\widetilde{O}(1)$ bits. Actually, this memory overhead can be avoided by labeling the paths leading to neighboring regions in the preprocessing phase without time penalty. Since the overhead is negligible we omit the details.

- Seeing that each worker in Theorem 3 has to store the final state of only one node in its region, it is not difficult to prove that the memory size needed for marking the nodes is $\widetilde{O}(\ell)$ bits, where ℓ is the maximum size (in bits) of the output labels.

Thus, assuming that ℓ is small compared to m (which is the case of many distributed algorithms), the memory size needed by a worker is dominated by the size of concatenated messages. Nevertheless, we remark that if a node in the original message passing algorithm sends the *same* message to all its neighbors then a worker can store only one message for that node in its internal memory, i.e., it does not need to store the same message many times. More generally, the memory size can be drastically improved for algorithms having particular behaviors. For instance, consider a message passing algorithm such that at each round a node *(i)* sends the same message M_1 to a bounded number of neighbors and *(ii)* the same message M_2 to the other neighbors. Then, it is not difficult to see that we can modify our concatenation mechanism so that the memory size needed by a worker is at most $\widetilde{O}(n \cdot b)$ bits. In particular, we have

Theorem 6 *For any n-node graph G, using n mobile agents with n^ϵ initial local view and having $\widetilde{O}(n)$ bit memory, a BFS tree with respect to a given node can be computed by a deterministic algorithm in $\widetilde{O}(D)$ time.*

Proof. Consider the following flood-like (message passing) algorithm: initially a root node begins the BFS computation by sending a $<$*Wave*$>$ message to its neighbors. Each node not yet in the tree waits upon the reception of $<$*Wave*$>$ messages. Whenever some $<$*Wave*$>$ messages are received for the first time at a given pulse, a node *(i)* chooses its parent in the tree, *(ii)* sends a $<$*Wave*$>$ message to all its neighbors, and *(iii)* terminates. Clearly, this is a correct synchronous BFS tree algorithm detecting the local termination. Furthermore, a node always sends the same message to all its neighbors. Thus the previous discussion concerning the memory size needed by the agents applies for this algorithm.

Nevertheless, the agents simulating the above BFS tree algorithm will terminate without knowing that the tree was effectively computed. One solution is to add a convergecast stage from the leaves of the tree to the root and a broadcast stage from the root to the leaves. One can see that the convergecast stage requires that each node sends one message to its father in the tree at a given pulse. Thus, the memory size needed by the agents is $\widetilde{O}(n)$ bits. As for the broadcast stage, a node v may send the same message to an unbounded number of children requiring more than $\widetilde{O}(n)$ bits per agent. However, we remark that a master agent in a region containing some neighbors of node v, say $(u_1 \cdots u_d)$, can determine that v is the father of $(u_1 \cdots u_d)$. Thus, the only information that must be delivered to the master is: "v is broadcasting a message". Thus, one can prove that the memory size required by agents to detect that the tree was computed is still $\widetilde{O}(n)$ bits. □

Theorem 7 *For any n-node graph G, using n mobile agents with n^ϵ initial local view and having $\widetilde{O}(1)$ bit memory, a leader can be computed in $\widetilde{O}(D)$ time.*

Proof.(sketch) Actually, one can apply the following algorithm: *(i)* simulate algorithm PART *(ii)* compute the region having the highest identifier *(iii)* the center of that region is elected leader. Computing the highest region is then proved to need $\widetilde{O}(1)$ bit memory per agent. This can be done using a 'wave-like' algorithm on the graph induced by the regions and started in parallel by the master of each region. □

Using the leader election, the BFS tree, the broadcast/convergecast process as basic procedures, one can think of other applications to more sophisticated algorithms.

8 Open problems

The results presented in this paper aim at understanding the power of computing with mobile agents having a small amount of knowledge. We believe that these results are a first step and an opening door towards new challenging problems in mobile agent computing. Hereafter, we give some open challenging questions:

1. Is a $\widetilde{O}(1)$ initial local view for the agents sufficient to compute as fast as with message passing? As regards to our randomized universal algorithm, this seems to be true (up to polylogarithmic factor) for many topologies and for many distributed tasks even when agents have no initial local view at all.

2. What lower bounds on the time ratio can we provide in the case of k mobile agents equipped with some limited initial local view? In particular, it is still not clear whether there is a $\sqrt{k}$ gap to close in our upper bound.

3. Is there any non trivial upper bound (or lower bound) on the time complexity if we constrain both the agent memory and the whiteboards to have limited capacity ? Actually, some of our techniques may help deriving new results concerning the memory of agents. However, it seems that new techniques have to be derived for studying computation models where the capacity of the whiteboards are limited.

References

[ABRS99] Baruch Awerbuch, Margrit Betke, Ronald L. Rivest, and Mona Singh. Piecemeal graph exploration by a mobile robot. *Information and Computation*, 152(2):155–172, 1999.

[AGLP89] B. Awerbuch, A. V. Goldberg, M. Luby, and S. A. Poltkin. Network decomposition and locality in distributed computation. In 30^{th} *IEEE Symp. on Found. of C. Sc. (FOCS)*, pages 364–369, 1989.

[AGVP90] Baruch Awerbuch, Oded Goldreich, Ronen Vainish, and David Peleg. A trade-off between information and communication in broadcast protocols. *J. ACM*, 37(2):238–256, 1990.

[AMZ06] Shehla Abbas, Mohamed Mosbah, and Akka Zemmari. Distributed computation of a spanning tree in a dynamic graph by mobile agents. In *IEEE Intl. Conf. on Engineering of Intelligent Systems*, pages 1–6, 2006.

[AOSY99] H. Ando, Y. Oasa, I. Suzuki, and M. Yamashita. A distributed memoryless point convergence algorithm for mobile robots with limited visibility. *IEEE Trans. Robot. Autom.*, 15(5):818–828, 1999.

[AP90] Baruch Awerbuch and David Peleg. Sparse partitions. In 31^{th} *IEEE Symp. on Found. of C. Sc. (FOCS)*, pages 503–513, 1990.

[AR93] Yehuda Afek and Moty Ricklin. Sparser: a paradigm for running distributed algorithms. *J. Algorithms*, 14(2):316–328, 1993.

[ASM95] H. Ando, I. Suzuki, and M.Yamashita. Formation and agreement problems for synchronous mobile robots with limited visibility. In *IEEE Symp. Intelligent Control*, pages 453–460, 1995.

[Awe85] Baruch Awerbuch. Complexity of network synchronization. *J. ACM*, 32(4):804–823, 1985.

[BBCD02] Fatima Belkouch, Marc Bui, Liming Chen, and Ajoy K. Datta. Self-stabilizing deterministic network decomposition. *J. Paral. and Dist. Computing*, 62(4):696–714, 2002.

[BFFS03] Lali Barrière, Paola Flocchini, Pierre Fraigniaud, and Nicolas Santoro. Can we elect if we cannot compare? In 15^{th} *ACM Symp. on Paral. Algo. and Arch. (SPAA'03)*, pages 324–332, 2003.

[BFFS07] Lali Barriere, Paola Flocchini, Pierre Fraigniaud, and Nicola Santoro. Rendezvous and election of mobile agents: Impact of sense of direction. *Th. of Comp. Sys. (ToCS)*, 40(2):143–162, 2007.

[BFNV06] Lelia Blin, Pierre Fraigniaud, Nicolas Nisse, and Sandrine Vial. Distributed chasing of network intruders. In 13^{th} *Colloquium on Structural Information and Communication Complexity (SIROCCO'06)*, pages 70–84, 2006.

[BS94] Michael A. Bender and Donna K. Slonim. The power of team exploration: two robots can learn unlabeled directed graphs. In 35^{th} *IEEE Symp. on Found. of C. Sc.*, pages 75–85, 1994.

[CGMO06] J. Chalopin, E. Godard, Y. Métivier, and R. Ossamy. Mobile agent algorithms versus message passing algorithms. In 10^{th} *Int Conf. On Princ. Of Dist. Sys.*, pages 187–201, 2006.

[CGP06] Jurek Czyzowicz, Leszek Gąsieniec, and Andrzej Pelc. Gathering few fat mobile robots in the plane. In 10^{th} *Intl. Conf. On Principles Of Dist. Sys.*, pages 350–364, 2006.

[CKR06] Colin Cooper, Ralf Klasing, and Tomasz Radzik. Searching for black-hole faults in a network using multiple agents. In 10^{th} *Intl. Conf. on Principles of Dist. Sys.*, pages 320–332, 2006.

[DFNS05] Shantanu Das, Paola Flocchini, Amiya Nayak, and Nicola Santoro. Distributed exploration of an unknown graph. In 12^{th} *Colloquium on Structural Information and Communication Complexity*, pages 99–114, 2005.

[DFNS06] Shantanu Das, Paola Flocchini, Amiya Nayak, and Nicola Santoro. Effective elections for anonymous mobile agents. In 17^{th} *Intl. Symp. on Algo. and Computation (ISAAC)*, pages 732–743, 2006.

[DFSY07] Shantanu Das, Paola Flocchini, Nicola Santoro, and Masafumi Yamashita. Fault-tolerant simulation of message-passing algorithms by mobile agents. In 14^{th} *Colloquium on Structural Information and Communication Complexity*, pages 289–303, 2007.

[DG06] Bilel Derbel and Cyril Gavoille. Fast deterministic distributed algorithms for sparse spanners. In 13^{th} *Colloquium on Structural Information and Communication Complexity*, pages 100–114, 2006.

[DP04] Anders Dessmark and Andrzej Pelc. Optimal graph exploration without good maps. *Theor. Comput. Sci.*, 326(1-3):343–362, 2004.

[FBDC99] Munehiro Fukuda, Lubomir F. Bic, Michael B. Dillencourt, and Jason M. Cahill. Messages versus messengers in distributed programming. *J. Parallel and Dist. Computing*, 57(2):188–211, 1999.

[FIP06a] Pierre Fraigniaud, David Ilcinkas, and Andrzej Pelc. Oracle size: a new measure of difficulty for communication tasks. In *PODC*, pages 179–187, 2006.

[FIP06b] Pierre Fraigniaud, David Ilcinkas, and Andrzej Pelc. Tree exploration with an oracle. In *MFCS*, pages 24–37, 2006.

[FKL07] Pierre Fraigniaud, Amos Korman, and Emmanuelle Lebhar. Local mst computation with short advice. In *Proceedings of the nineteenth annual ACM symposium on Parallel algorithms and architectures*, pages 154–160, 2007.

[FNS07] Paola Flocchini, Amiya Nayak, and Arno Schulz. Decontamination of arbitrary networks using a team of mobile agents with limited visibility. In 6^{th} *IEEE/ACIS Intl. Conf. on Computer and Information Science*, pages 469–474, 2007.

[FPSW05] Paola Flocchinia, Giuseppe Prencipe, Nicola Santoro, and Peter Widmayer. Gathering of asynchronous robots with limited visibility. *Theoretical Computer Science*, 337(1-3):147–168, 2005.

[FS06] Paola Flocchini and Nicola Santoro. Distributed security algorithms by mobile agents. In 8^{th} *Intl. Conf. Dist. Computing and Networking*, pages 1–14, 2006.

[IKK04] V. Isler, S. Kannan, and S. Khanna. Randomized pursuit-evasion with limited visibility. In *ACM-SIAM Symp. on Discrete Algorithms*, pages 1053–1063, 2004.

[KA02] Giorgos D. Kazazakis and Antonis A. Argyros. Fast positioning of limited-visibility guards for the inspection of 2d workspaces. In *IEEE Intl. Conf. on Intelligent Robots and Sys.*, pages 2843–2848, 2002.

[KK07] Amos Korman and Shay Kutten. Labeling schemes with queries. In 14^{th} *International Colloquium of Structural Information and Communication Complexity (SIROCCO)*, pages 109–123, 2007.

[KMP07] Ralf Klasing, Euripides Markou, and Andrzej Pelc. Gathering asynchronous oblivious mobile robots in a ring. *Theoretical Computer Science*, 2007. to appear.

[KMW04] Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. What cannot be computed locally! In 23^{rd} *ACM Symposium on the Principles of Distributed Computing (PODC), St. John's, Newfoundland, Canada*, July 2004.

[KMW06] Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. The price of being near-sighted. In 17^{th} *ACM-SIAM Symposium on Discrete Algorithms (SODA) Miami, Florida, USA*, January 2006.

[Kor] Amos Korman. General compact labeling schemes for dynamic trees. *accepted in Distributed Computing.*

[KP98] Shay Kutten and David Peleg. Fast distributed construction of small k-dominating sets and applications. *J. Algo.*, 28(1):40–66, 1998.

[KPR02] Amos Korman, David Peleg, and Yoav Rodeh. Labeling schemes for dynamic tree networks. In *Proceedings of the 19th Annual Symposium on Theoretical Aspects of Computer Science*, pages 76–87, 2002.

[Lin92] Nathan Linial. Locality in distributed graphs algorithms. *SIAM Journal on Computing*, 21(1):193–201, 1992.

[MS00] Shlomo Moran and Sagi Snir. Simple and efficient network decomposition and synchronization. *Theoretical Computer Science*, 243(1-2):217–241, 2000.

[NS95] Moni Naor and Larry Stockmeyer. What can be computed locally? *SIAM J. Comput.*, 24(6):1259–1277, 1995.

[PBD+04] Lei Pan, Lubomir F. Bic, Michael B. Dillencourt, Javid J. Huseynov, and Ming Kin Lai. Distributed parallel computing using navigational programming. *J. Parallel Programming*, 32(1):1–37, 2004.

[Pel00] David Peleg. *Distributed Computing: A Locality-Sensitive Approach.* SIAM Monographs on Disc. Math. & Appli., 2000.

[PS96] Alessandro Panconesi and Aravind Srinivasan. On the complexity of distributed network decomposition. *J. Algo.*, 20(2):356–374, 1996.

[RDW05] Malvika Rao, Gregory Dudek, and Sue Whitesides. Minimum distance localization for a robot with limited visibility. In *IEEE Intl. Conf. on Robot. and Automation*, pages 2439–2445, 2005.

[San01] Nicola Santoro. Distributed computations by autonomous mobile robots. In 28^{th} *Intl. conf. on current trends in theory and practice of c. sc.*, pages 110–115, 2001.

[SDY06] Samia Souissi, Xavier Défago, and Masafumi Yamashita. Using eventually consistent compasses to gather oblivious mobile robots with limited visibility. In 8^{th} *Intl. Symp. on Stabilization, Safety, and Sec.of Dist. Sys.*, pages 484–500, 2006.

Centre de recherche INRIA Futurs : Parc Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399